Predstavitev števil

r ... baza
$b = (b_n - 1, ..., b_0)$ ... število v bazi r s števkami b_1

- **nepredznačeno celo število** $V = \sum_{i=0}^{n} b_i r^i$
- **predznak in velikost (PV)** $V = (-1)^{b_{n-1}} \sum_{i=0}^{n-2} b_i 2^i$
  prvi bit (b_{n-1}) predstavlja predznak ($1 \to -, 0 \to +$), ostali pa velikost.
- **z odmikom** $V = \sum_{i=0}^{n-1} - \underbrace{2^{n-1}}_{\text{odmik}}$
- **eniški komplement (1'K)** $V = \sum_{i=0}^{n-1} b_i 2^i - b_{n-1}(2^n - 1)$
  prvi bit predstavlja predznak ($1 \to -, 0 \to +$). Negativno število dobimo tako, da invertiramo vse bite pozitivnega števila.
- **dvojiški komplement (2'K)** $V = \sum_{i=0}^{n-1} b_i 2^i - b_{n-1} 2^n$
  Negativno število dobimo tako da bite inveritramo in prištejemo 1.
- **pozicijska notacija** $V = \sum_{i=-m}^{n-1} b_i r^i$

Plavajoča vejica

- enojna natančnost *32 bit*
  prvi bit je predznak S ($1 \to -, 0 \to +$)
  sledi 8-biten eksponent e zapisan z odmikom ($e = E - 127$)
  sledi 23-bitna mantisa m
  prvi bit mantise je implicitno podan: če je eksponent $E = 0$ (v tem primeru je $e = -126$ in ne -127) je mantisa oblike $0, m$, sicer je oblike $1, m$.
  če so vsi biti eksponenta 1 in vsi biti mantise 0, je vrednost $\pm\infty$.
  če so vsi biti eksponenta 1 in vsi biti mantise niso 0, je vrednost NaN.
- dvojna natančnost *64 bit*
  prvi bit je predznak
  sledi 11-biten eksponent z odmikom 1023
  in 53-bitna mantisa
- normirana vrednost $(-1)^S \cdot (1, m) \cdot 2^e$
- denormirana vrednost $(-1)^S \cdot (0, m) \cdot 2^{e+1}$
- aritmetika v plavajoči vejici
  Zaokrožujemo k najbližji vrednosti, ki se jo da predstaviti (preferiramo soda števila).
  Pri računanju mantiso podaljšamo za 3 bite (*varovalni, zaokroževalni, lepljivi*)

Seštevanje v plavajoči vejici

- Obe števili zapišemo z večjim eksponentom (premik mantise, če izpadajo kake enice, se shranijo v leplijvem bitu)
- Mantisi seštejemo, če se pojavi prenos naprej, zmanjšamo mantiso in povečamo eksponent.

Množenje v plavajoči vejici

- eksponenta seštejemo
- mantisi zmnožimo v fiksni vejici

Prenos se pojavi, ko rezultat neke operacije preseže obseg števil. (nanaše se le na operacije z *nepredznačenimi* števili; pri 2'K se ignorira)
Preliv se pojavi če ima rezultat drugačen predznak kot števili.

CPE

t_{CPE} ... dolžina med dvema periodama
$f_{CPE} = 1/t_{CPE}$ ... frekvenca ure
$CPI = \sum_{i=0}^{n} CPI_i p_i$ ... povprečno število urinih preiod na ukaz
$MIPS = 1/(CPI \cdot t_{CPE} \cdot 10^6)$ ... million instructions per second

Predpomnilnik

t_{ap} ... čas dostopa do predpomnilnika
t_{ag} ... čas dostopa do glavnega pomnilnika
H ... verjetnost zadetka
t_B ... čas za prenos celega bloka
$t_a = t_{ap} + (1 - H) t_B$ ... povprečen čas dostopa

Set asociativni predpomnilnik (SAPP)

n ... dolžina naslova
$B = 2^b$ ... Število besed v bloku
$S = 2^s$ ... število setov, vsak set je majhen APP
$E = 2^e$ ... število blokov v setu (**stopnja asociativnosti**)
$M_b = S \cdot E$ ... število blokov v PP
$M = S \cdot E \cdot B$ ... velikost PP

- vsak naslov (A_i) iz GP se lahko preslika le v en set (S_i).
  $S_i = \underbrace{A_i[n-1:b]}_{\text{zgornji biti naslova}} \ \%2^s$ ... naslov seta *(predpomnilniški indeks)*
- iz naslova (A_i) se prebere naslov bloka (zgornjih $n - b$ bitov) in naslov besede (spodnjih b bitov).

Čisti asociativni predpomnilnik (APP)

- samo en set $S = 1$, posledično $E = M_b$
- vsak blok sprejme katerikoli del iz GP
- največja verjetnost zadetka

Direktni predpomnilnik

To je SAPP kjer je v vsakem setu le en blok $E = 1$.

Zgrešitve

Ob zgrešitvi se v PP prenese cel nov blok. Če v setu ni več prostora, se en blok zamenja (naključna strategija, *Least Recently Used* strategija).

Vrste zgrešitev

- **obvezne zgrešitve** pojavijo se vsakič, ko nek blok pomnilnika zahtevamo *prvič*.
- **velikostne zgrešitve** ko moramo naložiti nek blok, ki smo ga sicer že imeli a smo ga morali zamenjati zaradi prostorske stiske.
- **konfliktne zgrešitve** ko moramo naložiti nek blok, ki smo ga sicer že imeli a smo ga morali zamenjati, ker se je nek drug blok preslikal v isti set.

Vpliv predpomnilnika na hitrost

$$CPE_{\text{čas}} = (CPE_{\text{periode}_{\text{izvrš.}}} + CPE_{\text{periode}_{\text{čaka.}}}) t_{CPE}$$
$$CPE_{\text{periode}_{\text{izvrš.}}} = I \cdot CPI_{\text{idealni}}$$
$$CPE_{\text{periode}_{\text{čaka.}}} = N(1 - H) K_Z$$

I ... št. ukazov
N ... št. pomnilniških dostopov
H ... povprečna verjetnost zadetka PP
K_Z ... povprečna zgrešitvena kazen
CPI_{idealni} ... št. period na ukaz s predpostavko, da ni zgrešitev

Sklad

r0	ničla
r1-r23	splošno namenski registri
r24	prvi parameter ob klicu podprograma
r25	drugi parameter ob klicu podprograma
r26	bazni register za dolge skoke
r27	bazni register za dolge klice
r28	return value
r29	**FP** frame pointer
r30	**SP** stack pointer
r31	return address

```
push reg:
    sw 0(r30), reg
    subui r30, r30, #4
```

```
pop reg:
    addui r30, r30, #4
    lw reg, 0(r30)
```

Klic podprograma

- prva dva parametra shranimo v r24 in r25
- ostale parametre porinemo na sklad
- pokličemo podprogram z ukazom

Klicani podprogram ob vstopu

- na sklad porine **povratni naslov** (push r31)
- sa sklad porine **star kazalec na okvir** (push r29)
- **kazalec na okvir** nastavi na **skladovni kazalec** (r29 ← r30)
- po potrebi rezervira prostor na skladu za lokalne spremenljivke (skladovni kazalec premakne/pomanjša za velikost lokalnih spremenljivk)
- na sklad shrani vse registre, ki jih bo spreminjal

Klicani program med izvajanjem

- parametrov in lokalnih spremenljivk dostopamo prek kazalca na okvir
  - prva lokalna spremenljivka je na MEM[FP]
  - star kazalec na okvir je na MEM[FP+4]
  - povratni naslov je na MEM[FP+8]
  - zadnji parameter je na MEM[FP+12]

Klicani podprogram pred izstopom

- v r28 shrani vrednost, ki jo vrača
- s sklada obnovi vse shranjene registre
- s sklada pobriše vse lokalne spremenljivke (r30 ← r29)
- s sklada obnovi staro vrednost kazalca na okvir (pop r29)
- s sklada obnovi povratni naslov (pop r31)
- skoči na povratni naslov

Po vrnitvi iz podprograma

- izbrišemo parametre iz sklada

Cevovod

Procesor HIP deluje v 5 stopnjah (ki se odvijajo naenkrat)

- **IF** instruction fetch - *prevzem instrukcije iz GP*
- **ID** instruction decode - *dekodiranje ukaza in branje iz registrov*
- **EX** execute - *izvršitev operacije (na ALU)*
- **MEM** memory - *dostop do pomnilnika*
- **WB** write back - *shranjevanje reultata (v reg)*

Cevovodne nevarnosti

Strukturne nevarnosti

Več stopenj uporablja isto enoto. *ne pri HIP*

Podatkovne nevarnosti

Ukaz kot parameter potrebuje še neizračunan rezultat prejšnjega ukaza *samo v ID*. Rešujemo jih lahko:

- **programsko** - vstavljanje nop za problematične ukaze
- **cevovodna zaklenitev** - ko procesor (v stopnji ID) zazna nevarnost, v cevovod vstavlja "mehurčke"
- **premoščanje** - rezultate iz EX, MEM ali WB prenesemo v ID, če to ni mogoče (pri load), vstavimo mehurček
- **razvrščanje** - ukaze razvrstimo tako, da odpravimo nevarnost

Kontrolne nevarnosti

Pri ukazih (skokih), ki spreminjajo PC.
HIP nov naslov izračuna že v ID, pri pogojnih skokih se PC lahko spremeni v EX (vsebina IF in ID postane neveljavna) Procesor predpostavi, da skoka ne bo.
Rešitve:

- **vstavljanje mehurčkov** v primeru skoka se v stopnji IF in ID vstavita mehurčka, sicer pa gre vse normalno naprej.
- predikcija skočnega pogoja
  - **statična** med izvrševanjem se ne spreminja
  - **dinamična** procesor si zapomni vrednost prejšnjih skokov
- **zakasnjeni skoki** : v *skočne reže* (pri HIP 2 ukaza za skokom) damo ukaze, ki bi se morali izvesti pred skokom in ne vplivajo na pogoj skoka.